Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

Course: CH-232-A
Date: 2019-10-18
Due: 2019-10-25

ICS 2019 Problem Sheet #6

Problem 6.1: *character encoding* (1+1+1 = 3 points)

a) The hexadecimal numbers shown below represent ASCII character code points. What was the original ASCII text? (Pay attention to any control codes you may find.)

```
53 69 6d 70 6c 69 63 69 74 79 2c 20 63 61 72 72
69 65 64 20 74 6f 20 74 68 65 20 65 78 74 72 65
6d 65 2c 20 62 65 63 6f 6d 65 73 20 65 6c 65 67
61 6e 63 65 2e 0a 2d 20 4a 6f 6e 20 46 72 61 6e
6b 6c 69 6e 0a
```

b) The hexadecimal numbers shown below represent UTF-8 encoded characters.

```
7c 7b 2d 7d e2 88 aa 7b e2 88 92 7d e2 88 aa 7b
c2 ad 7d e2 88 aa 7b e2 80 93 7d e2 88 aa 7b e2
80 91 7d e2 88 aa 7b e2 80 94 7d e2 88 aa 7b e2
80 92 7d 7c 20 3d 20 37 0a
```

For every UTF-8 code point, write down the corresponding Unicode code point and the associated Unicode character name. Fill out a table that starts like this:

UTF-8	Unicode	Name
7c	U+007C	VERTICAL LINE
⋮	⋮	⋮

Pay attention to any control codes you may find.

c) The unified Chinese, Japanese, Korean and Vietnamese characters have code points in the CJK Unified Ideographs code block (U+4E00 until U+9FFF) in the Basic Multilingual Plane (plane 0). Consider a Chinese text with 800 000 characters. How many (8-bit) bytes does the representation of this text use in UTF-32 and UTF-8 format?

Problem 6.2: *date and time calculations* (1+1+1 = 3 points)

a) Which of the following dates equivalent? Explain why or why not.

No.	Date and Time
1	2019-10-15T15:15:00+02:00
2	2019-10-13T17:15:00+00:00
3	2019-10-13T13:15:00+00:00
4	2019-10-13T15:15:00-02:00
5	2019-10-13T00:30:00-12:45
6	2019-10-14T05:15:00+12:00

b) RFC 3339 allows time zone offsets such as -00:00. Why is this useful?

c) What is the “year 2038 problem”? How can it be solved?

Problem 6.3: *emoji substitution cipher (haskell)* (2+2 = 4 points)

The Unicode character set contains a number of emojis and some people use them extensively. Perhaps there is a deeper meaning hidden in these emojis...

a) Implement a function `enc :: String -> String` that converts all lowercase ASCII letters into emojis and all uppercase ASCII letters into symbols representing animals.

b) Implement a function `dec :: String -> String` that implements the inverse function, converting emojis and symbols representing animals into ASCII letters.

The emojis start at the code point U+1f600 and the animal symbols start at the code point U+1f400. Feel free to use the functions defined in `Data.Char` for conversion and for testing to which character class a given character belongs.

Below is a `Main.hs` module that reads text from the standard input and calls the `enc` or `dec` function depending on whether the input includes ASCII letters or not.

```haskell
{- |
   Module: Main.hs
-}

import System.IO
import Data.Char
import Emoji

-- Peek into the content to decide whether we encode or decode.
convert :: String -> String
convert xs
  | null $ filter (\c -> isLetter c && isAscii c) xs = dec xs
  | otherwise                                        = enc xs

main = do
  contents <- getContents
  putStr $ convert contents
```

Submit your Haskell code as a plain text file.